

## FIGURE 1

[illegible]

## FIGURE 2

MSNQFAGHQTSFCATEDYTSYCWYIDEPQGGEEIQEAGEVPSCHTSIDLCYTHACLASLS  
 IIAVLELLAHIAVEEQIWFIDCVGEFPGHLESPVDFLAGDEIFAVPAAVFMVLLSNICLLLEP  
 EDALPELTLASAPESQDCKTEAFPCAWEILGLETTYAALYYPLAACATAGHTAAHLLGSTLS  
 WAHLGGVQVWQPARCTQVFKIYKYYSLLASLELLLCGLGFLSLWYFVQLVPSFEDERTGAGSK  
 GLQSNYSSEYILEHLLCEKELGSSYHTSEHGFLSWARVCLPHCIYTPQPGHLLPDLVLISA  
 TLTGTAIYQVALLLVGVVETIQKVRAGVTTDVSYLLAGEGIVLSEDEQEVVELVFNHLLW  
 ALEVCIYISALVLSCLLTFELVLMPSLVTHRTNDRALHRCGAALDISPLHRSHPHPRQATFCW  
 MSESAYQTAFICLGLLVQQLIEFLGTTALAEFLVLMPLHGRNILLERSLESSWPFWLTLA  
 LAVILQDMAAHWVLETHDCHPQLTNERVLYAATELLEPLNVLVGAMVATWRVLLSALYN  
 AIIHLCQMLSLLEPPAATLDPGYYTYRNFELKIEVSQSHDAMTAFCSLLLOAQGLLPRMA  
 APQDSLPEGEEDECMQLLQTKDSMAKGARPGAGEGPARWGLAYTLIDHNPTLQVFKTALL  
 GANGAQP

Important features of the protein:

Signal peptide:

None

Transmembrane domain:

54-69  
 102-119  
 148-166  
 207-222  
 301-320  
 364-380  
 431-451  
 474-489  
 560-535

Motif file:

Motif name: N-glycosylation site.

8-12

Motif name: N-myristoylation site.

50-56  
 176-182  
 241-247  
 317-323  
 341-347  
 525-531  
 627-633  
 631-637  
 640-646  
 661-667

Protein Data Bank accession number: PDB entry: 1A01

Motif name: N-glycosylation site. Motif type: N-glycosylation site.

132-140

## FIGURE 3A

PRO XXXXXXXXXXXXXXXX (Length = 15 amino acids)

Comparison Protein XXXXXYYYYYYY (Length = 12 amino acids)

% amino acid sequence identity =

(the number of identically matching amino acid residues between the two polypeptide sequences as determined by ALIGN-2) divided by (the total number of amino acid residues of the PRO polypeptide)

5 divided by 15 = 33.3%

## FIGURE 3B

PRO XXXXXXXXXXXX (Length = 10 amino acids)

Comparison Protein XXXXXYYYYYYYZZYZ (Length = 15 amino acids)

5,

% amino acid sequence identity

(the number of identically matching amino acid residues between the two polypeptide  
sequences as determined by ALIGN 2) divided by (the total number of amino acid residues  
10 of the PRO polypeptide)

5 divided by 10 = 50%

## FIGURE 3C

PRO DNA	NNNNNNNNNNNNNNNN	(Length	14
nucleotides)			
5 Comparison DNA	NNNNNNNLLLLLLLLL	(Length	16
nucleotides)			

% nucleic acid sequence identity --

10 (the number of identically matching nucleotides between the two nucleic acid sequences as determined by ALIGN 2) divided by (the total number of nucleotides of the PRO DNA nucleic acid sequence)

6 divided by 14 = 42.9%

15

## FIGURE 3D

PRO DNA	NNNNNNNNNNNNNN	(Length = 12 nucleotides)
Comparison DNA	NNNNILLVV	(Length = 9 nucleotides)

% nucleic acid sequence identity =

(the number of identically matching nucleotides between the two nucleic acid sequences as determined by ALIGN 2) divided by (the total number of nucleotides of the PRO DNA nucleic acid sequence)

4 divided by 12 = 33.3%

# FIGURE 4A

```

/*
 *
 * C C increased from 12 to 15
 * Z is average of I-Q
 * B is average of ND
 * match with stop is -M, stop stop = 0, J (joker) match = 0
 */
#define M 8 /* value of a match with a stop */

10 int day[26][26] = {
/* A */ { 2, 0, 2, 0, 0, 4, 1, 1, 1, 0, -1, -2, -1, 0, M, 1, 0, 2, 1, 1, 0, 0, 6, 0, 3, 0},
/* B */ { 0, 3, 4, 3, 2, -5, 0, 1, 2, 0, 0, 3, 2, 2, M, 1, 1, 0, 0, 0, 0, -2, -5, 0, 3, 1},
15 /* C */ { 2, 4, 15, 5, 5, 4, 3, 3, 2, 0, 5, 6, 5, 4, M, 3, 5, 4, 0, 2, 0, 2, 8, 0, 0, 5},
/* D */ { 0, 3, 5, 4, 3, 6, 1, 1, 2, 0, 0, 4, 3, 2, M, 1, 2, 1, 0, 0, 0, 2, -7, 0, 4, 2},
/* E */ { 0, 2, 5, 3, 4, 5, 0, 1, 2, 0, 0, 3, 2, 1, M, 1, 2, 1, 0, 0, 0, 2, -7, 0, 4, 3},
/* F */ { 4, 5, 4, 6, 5, 9, 5, 2, 1, 0, -5, 2, 0, 4, M, 5, 5, 4, 3, 3, 0, 1, 0, 0, 7, 5},
/* G */ { 1, 0, 3, 1, 0, 5, 5, 2, 3, 0, -2, 4, 3, 0, M, 1, 1, 3, 1, 0, 0, 1, -7, 0, 5, 0},
20 /* H */ { 1, 1, 3, 1, 1, 2, 2, 6, 2, 0, 0, 2, 2, 2, M, 0, 3, 2, 1, 1, 0, 2, -3, 0, 0, 2},
/* I */ { 1, 2, 2, 2, 2, 1, 3, 2, 5, 0, 2, 2, 2, 2, M, 2, 2, 2, 1, 0, 0, 4, 5, 0, 1, 2},
/* J */ { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, M, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
/* K */ { 1, 0, 5, 0, 0, 5, 2, 0, 2, 0, 5, 3, 0, 1, M, 1, 1, 3, 0, 0, 0, 2, -3, 0, 4, 0},
/* L */ { 2, 3, 6, 4, 3, 2, 4, 2, 2, 0, 3, 6, 4, 3, M, 3, 2, 3, 3, 1, 0, 2, -2, 0, 1, 2},
25 /* M */ { 1, 2, 5, 3, 2, 0, 3, 2, 2, 0, 0, 4, 6, -2, M, 2, 1, 0, 2, 1, 0, 2, -4, 0, 2, 1},
/* N */ { 0, 2, 4, 2, 1, 4, 0, 2, 2, 0, 1, 3, 2, 2, M, 1, 1, 0, 1, 0, 0, 2, -4, 0, 2, 1},
/* O */ { M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M},
0, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M},
/* P */ { 1, 1, 3, 1, 1, 5, 1, 0, 2, 0, 1, 3, 2, 1, M, 6, 0, 0, 1, 0, 0, 1, -6, 0, 5, 0},
30 /* Q */ { 0, 1, 5, 2, 2, 5, 1, 3, 2, 0, 1, 2, 1, 1, M, 0, 4, 1, 1, 1, 0, 2, -5, 0, 4, 3},
/* R */ { 2, 0, 4, 1, 1, 4, 3, 2, 2, 0, 3, 3, 0, 0, M, 0, 1, 6, 0, 1, 0, 2, -2, 0, 4, 0},
/* S */ { 1, 0, 0, 0, 0, 3, 1, 1, 1, 0, 0, 3, 2, 1, M, 1, 1, 0, 2, 1, 0, 1, -2, 0, 3, 0},
/* T */ { 1, 0, 2, 0, 0, 3, 0, -1, 0, 0, 0, 1, 1, 0, M, 0, 1, 1, 1, 3, 0, 0, 5, 0, 3, 0},
/* U */ { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, M, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
35 /* V */ { 0, 2, 2, 2, 2, 1, 1, 2, 4, 0, 2, 2, 2, 2, M, 1, 2, 2, 1, 0, 0, 4, -6, 0, 2, 2},
/* W */ { 6, 5, 8, 7, 7, 0, 7, 3, 5, 0, 3, 2, 4, 4, M, 6, 5, 2, 2, 5, 0, 6, 17, 0, 0, 6},
/* X */ { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, M, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
/* Y */ { 5, 3, 0, 4, 4, 7, 5, 0, 1, 0, 4, 1, 2, 2, M, 5, 4, 4, 3, 3, 0, 2, 0, 0, 10, 4},
40 /* Z */ { 0, 1, 5, 2, 3, 5, 0, 2, 2, 0, 0, 2, 1, 1, M, 0, 3, 0, 0, 0, 0, 2, -6, 0, 4, 4}
};

```

## FIGURE 4B

```

/*
 */
#include <stdio.h>
#include <ctype.h>

#define MAXJMP 16 /* max jumps in a diag */
#define MAXGAP 24 /* don't continue to penalize gaps larger than this */
#define JMPS 1024 /* max jmps in an path */
10 #define MX 4 /* save if there's at least MX-1 bases since last jmp */

#define DMAT 3 /* value of matching bases */
#define DMIS 0 /* penalty for mismatched bases */
15 #define DINS0 8 /* penalty for a gap */
#define DINS1 1 /* penalty per base */
#define FINS0 8 /* penalty for a gap */
#define FINS1 4 /* penalty per residue */

struct jmp {
20     short n[MAXJMP]; /* size of jmp (neg for del) */
     unsigned short x[MAXJMP]; /* base no. of jmp in seq x */
}; /* limits seq to 2^16-1 */

struct diag {
25     int score; /* score at last jmp */
     long effect; /* effect of prev block */
     short jmp; /* current jmp index */
     struct jmp ip; /* list of jmps */
30 };

struct path {
     int spc; /* number of leading spaces */
     short n[JMPS]; /* size of jmp (gap) */
     int x[JMPS]; /* loc of jmp (last elem before gap) */
35 };

char *ofile; /* output file name */
char *names[2]; /* seq names: getseqs() */
char *prog; /* prog name for err msgs */
40 char *seqs[2]; /* seqs: getseqs() */
int dmax; /* best diag: nw() */
int dmax0; /* final diag */
int dna; /* set of dna: mamo() */
int endgaps; /* set of penalizing end gaps */
45 int gapx, gapy; /* total gaps in seqs */
int len0, len1; /* seq lens */
int ngapx, ngapy; /* total size of gaps */
int smax; /* max score: nw() */
int *xbm; /* bitmap for matching */
50 long offset; /* current offset in jmp file */
struct diag *dx; /* holds diagonals */
struct path *pp[2]; /* holds path for seqs */

char *calloc(), *malloc(), *index(), *strcpy();

```



## FIGURE 4C

```

/* Needleman-Wunsch alignment program
 *
 * usage: prog file1 file2
 *   where file1 and file2 are two dna or two protein sequences
 *   The sequences can be in upper- or lower case and may contain ambiguity
 *   Any lines beginning with '#', '!' or '?' are ignored
 *   Max file length is 65535 (limited by unsigned short x in the jmp struct)
 *   A sequence with 1/3 or more of its elements ACGTU is assumed to be DNA
 *   Output is in the file "align.out"
 *
 * The program may create a tmp file in /tmp to hold info about traceback
 * Original version developed under BSD 4.3 on a vax 8650
 */

#include "nw.h"
#include "day.h"

static  dbval[26] = {
1,14,2,13,0,0,4,11,0,0,12,0,3,15,0,0,0,5,6,8,8,7,9,0,10,0
};

static  pval[26] = {
1, 2[(1<= (CD)'A')?(1<= (N)'A'), 4, 8, 16, 32, 64,
128, 256, 0xFFFFFF, 1<= 10, 1<= 11, 1<= 12, 1<= 13, 1<= 14,
1<= 15, 1<= 16, 1<= 17, 1<= 18, 1<= 19, 1<= 20, 1<= 21, 1<= 22,
1<= 23, 1<= 24, 1<= 25](1<= (F)'A')?(1<= (Q)'A')
};

main(ac, av)
int    ac;
char   *av[];
{
    prog = av[0];
    if (ac != 3) {
        fprintf(stderr, "usage: %s file1 file2\n", prog);
        fprintf(stderr, "where file1 and file2 are two dna or two protein sequences\n");
        fprintf(stderr, "The sequences can be in upper- or lower case\n");
        fprintf(stderr, "Any lines beginning with '#', '!' or '?' are ignored\n");
        fprintf(stderr, "Output is in the file \"align.out\"\n");
        exit(1);
    }
    names[0] = av[1];
    names[1] = av[2];
    seqx[0] = getseq(names[0], &len0);
    seqx[1] = getseq(names[1], &len1);
    dbm = (dna)* dbval - pval;

    endgaps = 0;
    ofile = "align.out";

    nw();
    readjumps();
    print();
}

```

main

## FIGURE 4D

```

/* do the alignment, return best score: main()
 * dna: values in Fitch and Smith, PNAS, 80, 1382-1386, 1983
 * pro: PAM 250 values
5  * When scores are equal, we prefer mismatches to any gap, prefer
 * a new gap to extending an ongoing gap, and prefer a gap in seqx
 * to a gap in seqy
 */
nw()
10 {
    char      *px, *py;      /* seqs and ptrs */
    int       *ndely, *dely; /* keep track of dely */
    int       *ndelx, *delx; /* keep track of delx */
    int       *tmp;          /* for swapping row0, row1 */
15  int       mis;           /* score for each type */
    int       ins0, ins1;    /* insertion penalties */
    register  id;            /* diagonal index */
    register  n;             /* jmp index */
    register  *col0, *col1;  /* score for curr, last row */
20  register  xx, yy;        /* index into seqs */

    dx = (struct diag *)g_calloc("to get dx", len0 + len1 + 1, sizeof(struct diag));

    ndely = (int *)g_calloc("to get ndely", len1 + 1, sizeof(int));
25  dely = (int *)g_calloc("to get dely", len1 + 1, sizeof(int));
    col0 = (int *)g_calloc("to get col0", len1 + 1, sizeof(int));
    col1 = (int *)g_calloc("to get col1", len1 + 1, sizeof(int));
    mis0 = (dna)? DINS0 - PINS0;
    mis1 = (dna)? DINS1 - PINS1;

30  smax = 10000;
    if (endgaps) {
        for (col0[0] = dely[0] = mis0, yy = 1; yy <= len1; yy++) {
            col0[yy] = dely[yy] = col0[yy-1] + mis1,
35  ndely[yy] = yy;
        }
        col0[0] = 0; /* Waterman Bull Math Biol 84 */
    }
    else
40  for (yy = 1; yy <= len1; yy++)
        dely[yy] = mis0;

    /* fill in match matrix
    */
45  for (px = seqx[0], xx = 1, xx <= len0; px++, xx++) {
        /* initialize first entry in col
        */
        if (endgaps) {
            if (xx == 1)
50  col1[0] = delx = (mis0 + ins1);
            else
                col1[0] = delx = col0[0] + mis1;
            ndelx = xx;
        }

        }
    }
60 }

```

nw

# FIGURE 4E

...NW

```

5  for (py = seqx[1], yy = 1, yv = 1; py != 0, yy != 0) {
    ms = col0[yv-1];
    if (dna)
        ms += (xbm[*px-'A']&xbm[*py-'A'])/DMAT+DMIS;
    else
        ms += dxy[*px-'A'][*py-'A'];

10  /* update penalty for del in x seq;
    * favor new del over ongoing del
    * ignore MAXGAP if weighting endgaps
    */
    if (endgaps || ndely[yv] < MAXGAP) {
15      if (col0[yv] - ms0 >= ndely[yv]) {
          dely[yv] = col0[yv] - (ms0 + ms1);
          ndely[yv] = 1;
      } else {
          dely[yv] = ms1;
          ndely[yv] += 1;
20      }
    } else {
        if (col0[yv] - (ms0 + ms1) >= ndely[yv]) {
            dely[yv] = col0[yv] - (ms0 + ms1);
            ndely[yv] = 1;
25      } else
          ndely[yv] += 1;
    }

30  /* update penalty for del in y seq;
    * favor new del over ongoing del
    */
    if (endgaps || ndelx < MAXGAP) {
        if (col1[yy-1] - ms0 >= ndelx) {
35      dely[yy] = col1[yy-1] - (ms0 + ms1);
          ndelx = 1;
        } else {
            dely[yy] = ms1;
            ndelx += 1;
40      }
    } else {
        if (col1[yy-1] - (ms0 + ms1) >= ndelx) {
            dely[yy] = col1[yy-1] - (ms0 + ms1);
            ndelx = 1;
45      } else
          ndelx += 1;
    }

50  /* pick the maximum score, we're favoring
    * ms over any del and delx over dely
    */
    r =

```

...BBV'

```

(void) free((char *)ndch);
(void) free((char *)ndch);
(void) free((char *)scd0); (void) free((char *)scd1);

```

## FIGURE 4G

```

/*
 *
 * print() -- only routine visible outside this module
 */
/*
 * static
 * getmat() -- trace back best path, count matches: print()
 * pr_align() -- print alignment of described in array p[]: print()
 * dumpblock() -- dump a block of lines with numbers, stars: pr_align()
10  * nums() -- put out a number line: dumpblock()
 * putline() -- put out a line (name, [num], seq, [num]): dumpblock()
 * stars() -- put a line of stars: dumpblock()
 * stripname() -- strip any path and prefix from a sequence
 */
15
#include "nw.h"

#define SPC 3
#define P_LINE 256 /* maximum output line */
20 #define P_SPC 3 /* space between name or num and seq */

extern day[26][26];
int olen; /* set output line length */
FILE *fx; /* output file */

25
print()
{
    int lx, ly, firstgap, lastgap; /* overlap */

    30 if ((fx = fopen(ofile, "w")) == 0) {
        fprintf(stderr, "%s: can't write (%s)\n", prog, ofile);
        cleanup(1);
    }
    fprintf(fx, " > first sequence: %s (length = %d)\n", namex[0], len0);
    35 fprintf(fx, " > second sequence: %s (length = %d)\n", namex[1], len1);
    olen = 60;
    lx = len0;
    ly = len1;
    firstgap = lastgap = 0;
    40 if (dmax < len1 + 1) { /* leading gap in x */
        pp[0].spe = firstgap - len1 - dmax + 1;
        ly -= pp[0].spe;
    }
    else if (dmax > len1 + 1) { /* leading gap in y */
    45 pp[1].spe = firstgap - dmax - (len1 + 1);
        lx -= pp[1].spe;
    }
    if (dmax0 < len0 + 1) { /* trailing gap in x */
        lastgap = len0 - dmax0 + 1;
        50 lx += lastgap;
    }
    else if (dmax0 > len0 + 1) { /* trailing gap in y */
        lastgap = dmax0 - (len0 + 1);
        55 ly += lastgap;
    }
}
60

```

print

## FIGURE 4H

```

/*
 * trace back the best path, count matches
 */
5 static
getmat(ix, iy, firstgap, lastgap)                                getmat
    int      ix, iy;                                           /* "core" (minus endgaps) */
    int      firstgap, lastgap;                                /* leading trailing overlap */
{
10     int      nm, i0, i1, siz0, siz1;
    char      outx[32];
    double     pct;
    register   n0, n1;
    register char *p0, *p1;

15     /* get total matches, score
     */
    i0 = i1 = siz0 = siz1 = 0;
    p0 = seqx[i0] + pp[1] spc;
    p1 = seqx[i1] + pp[0] spc;
20     n0 = pp[1] spc + 1;
    n1 = pp[0] spc + 1;

    nm = 0;
25     while ( *p0 && *p1 ) {
        if (siz0) {
            p1 ++;
            n1 ++;
            siz0 --;
30         }
        else if (siz1) {
            p0 ++;
            n0 ++;
            siz1 --;
35         }
        else {
            if (xbm[*p0-'A'] & xbm[*p1-'A'])
                nm ++;
            if (n0 ++ == pp[0].x[i0])
                siz0 = pp[0].n[i0 ++];
            if (n1 ++ == pp[1].x[i1])
                siz1 = pp[1].n[i1 ++];
            p0 ++;
            p1 ++;
40         }
45     }

    /* pct homology:
     * if penalizing endgaps, base is the shorter seq
     * else, knock off overhangs and take shorter core
     */
    if (endgaps)
        ix = (len0 + -len1)? len0 : len1;
    else
        ix = (ix + 1)? ix : 1x;

```

# FIGURE 4I

```

fprintf(fx, " < gaps in first sequence: %d", gapx);
if (gapx) {
    (void) sprintf(outx, " (%d %s%s)",
        gapx, (dna)? "base": "residue", (gapx > 1)? "" : "s");
    fprintf(fx, "%s", outx);

fprintf(fx, " < gaps in second sequence: %d", gapy);
if (gapy) {
    (void) sprintf(outx, " (%d %s%s)",
        gapy, (dna)? "base": "residue", (gapy > 1)? "" : "s");
    fprintf(fx, "%s", outx);
}
if (dna)
    fprintf(fx,
        "\n< score: %d (match = %d, mismatch = %d, gap penalty = %d + %d per base)\n",
        smax, DMAX, DMIS, DINSO, DINSI);
else
    fprintf(fx,
        "\n< score: %d (Dayhoff PAM 250 matrix, gap penalty = %d + %d per residue)\n",
        smax, PINSO, PINSI);
if (endgaps)
    fprintf(fx,
        "\n< endgaps penalized, left endgap: %d %s%s, right endgap: %d %s%s\n",
        firstgap, (dna)? "base": "residue", (firstgap > 1)? "" : "s",
        lastgap, (dna)? "base": "residue", (lastgap > 1)? "" : "s");
else
    fprintf(fx, "\n< endgaps not penalized\n");
}

static int nm; /* matches in core - for checking */
static int lmax; /* lengths of stripped file names */
static int ii[2]; /* jmp index for a path */
static int nc[2]; /* number at start of current line */
static int ni[2]; /* current elem number - for gapping */
static int sz[2];
static char *ps[2]; /* ptr to current element */
static char *po[2]; /* ptr to next output char slot */
static char out[2][P_LINE]; /* output line */
static char star[P_LINE]; /* set by stars() */

/*
 * print alignment of described in struct path pp[]
 */
static
pr_align()
{
    int nm; /* char count */
    int more;
    register int i;

    for (i = 0; lmax > 0; i < 2; i++) {
        nm = strlen(name[i]);
        if (nm > lmax)
            lmax = nm;

        ps[i] = n[i] = 0;
        po[i] = seq[i];
        pr[i] = out[i];
    }

```

...getmat

pr\_align

## FIGURE 4J

...pr align

```

5  for (nn = nm - 0; more = 1; more, ) {
6      for (i = more - 0; i < 2; i++) {
7          /*
8           * do we have more of this sequence?
9           */
10         if (!ps[i])
11             continue;
12
13         more++;
14
15         if (pp[i].spe) { /* leading space */
16             *pc[i]++ = ' ';
17             pp[i].spe--;
18         }
19         else if (siz[i]) { /* in a gap */
20             *pc[i]++ = ' ';
21             siz[i]--;
22         }
23         else { /* we're putting a seq element
24             */
25             *pc[i] = *ps[i];
26             if (islower(*ps[i]))
27                 *ps[i] = toupper(*ps[i]);
28             pc[i]++;
29             p[i]++;
30
31             /*
32              * are we at next gap for this seq?
33              */
34             if (m[i] == pp[i].x[b[i]]) {
35                 /*
36                  * we need to merge all gaps
37                  * at this location
38                  */
39                 siz[i] = pp[i].o[b[i]] + 1;
40                 while (m[i] == pp[i].x[b[i]])
41                     siz[i] += pp[i].o[b[i]] + 1;
42             }
43             m[i]++;
44         }
45     }
46     if (i + nm == olen || !more && nm) {
47         dumpblock();
48         for (i = 0; i < 2; i++)
49             pc[i] = out[i];
50         nm = 0;
51     }
52 }
53
54 /*
55  * dump a block of lines, including numbers, stars: pr align()
56  */
57
58 void dumpblock()
59 {
60     for (i = 0; i < 2; i++)
61         *pc[i] = "00";

```



# FIGURE 4K

...dumpblock

```

5      (void) puts("\n", fx);
      for (i = 0; i < 2; i++) {
          if (*out[i] && (*out[i] != ' ') || (*pc[i] != ' ')) {
              if (i == 0)
                  numstr(i);
              if (i == 0 && *out[1])
                  stars(i);
10         putline(i);
              if (i == 0 && *out[1])
                  fprintf(fx, star);
              if (i == 1)
                  numstr(i);
          }
      }
  }

20  /*
   * put out a number line: dumpblock()
   */
  static
  numstr(i)
25      int i; /* index in out[] holding seq line */
  {
      char nline[P * LINE];
      register int i;
      register char *pn, *px, *py;

30      for (pn = nline, i = 0, j = lmax + P * SPC, i++, pn++)
          *pn = ' ';
      for (i = 0; i < out[i]; i++) {
          if (*py == ' ') || (*py == ' ')
35              *pn = ' ';
          else {
              if (i % 10 == 0 || (i % 10 == 1 && out[i] != 1)) {
                  j = (i + 0) / 10;
                  for (px = pn; j / 10 > 0; px++)
40                      *px = j % 10 + '0';
                  if (i < 0)
                      *px = '-';
              }
              else
45                  *pn = ' ';
              i++;
          }
      }
      *pn = '\0';
      out[i] = i;
      for (pn = nline; *pn; pn++)
          (void) puts(*pn, fx);
      (void) puts("\n", fx);
50  }

```

numstr

putline

# FIGURE 4L

...putline

```

5      int          i;
      register char *px;

      for (px = namex[ix], i = 0; *px && *px != '\0'; px++, i++)
          (void) putc(*px, fx);
10     for (i = lmax + P - SPC; i > 0; i--)
          (void) putc(' ', fx);

      /* these count from 1:
       * m[] is current element (from 1)
       * nc[] is number at start of current line
       */
15     for (px = out[ix], *px; px > 0; px++)
          (void) putc(*px & 0x7F, fx);
      (void) putc('\n', fx);
20 }

/*
 * put a line of stars (seqs always in out[0], out[1]): dumpblock()
 */
25 static
stars()
{
      int          i;
      register char *p0, *p1, cx, *px;

30     if (!out[0] || (*out[0] == '\0' && *p0[0] == '\0') ||
        !out[1] || (*out[1] == '\0' && *p1[1] == '\0'))
          return;

      px = star;
35     for (i = lmax + P - SPC; i > 0; i--)
          *px++ = ' ';

      for (p0 = out[0], p1 = out[1], *p0 && *p1; p0++, p1++) {
40         if (isalpha(*p0) && isalpha(*p1)) {
              if (xbm[*p0 - 'A'] & xbm[*p1 - 'A']) {
                  cx = '*';
                  nm++;
              }
45             else if (!dma && !day[*p0 - 'A'][*p1 - 'A'] > 0)
                  cx = ' ';
              else
                  cx = '?';
          }
50         else
              cx = '?';

          *px++ = cx;
      }
      *px++ = '\n';
60 }

```

stars

## FIGURE 4M

stripname

```
/*
 * strip path or prefix from pn, return len: pn - align0
 */
5 static
stripname(pn)
    char *pn; /* file name (may be path) */
{
10     register char *px, *py;
        py = 0;
        for (px = pn; *px; px++)
            if (*px == '/')
                py = px + 1;
15     if (py)
        (void) strcpy(pn, py);
        return(strlen(pn));
20 }
25
30
35
40
45
50
55
60
```

## FIGURE 4N

```

/*
 * cleanup() - cleanup any tmp file
 * getseq() - read in seq, set dna, len, maxlen
 * g_calloc() - calloc() with error checking
 * readpmps() - get the good pmps, from tmp file if necessary
 * writepmps() - write a filled array of pmps to a tmp file - nwc()
 */
#include "nw.h"
#include <sys/file.h>

char *pname = "/tmp/hompXXXXXX"; /* tmp file for pmps */
FILE *fp;

int cleanup(); /* cleanup tmp file */
long lseek();

/*
 * remove any tmp file if we blow
 */
cleanup(i)
int i;
{
    if (fp)
        (void) unlink(pname);
    exit(i);
}

/*
 * read, return ptr to seq, set dna, len, maxlen
 * skip lines starting with '!', '?', or '.'
 * seq in upper or lower case
 */
char *
getseq(file, len)
char *file; /* file name */
int *len; /* seq len */
{
    char line[1024]; *pseq;
    register char *px, *py;
    int natgc, tlen;
    FILE *fp;

    if ((fp = fopen(file, "r")) == 0) {
        fprintf(stderr, "%s: can't read '%s'\n", prog, file);
        exit(1);
    }
    tlen = natgc = 0;
    while (fgets(line, 1024, fp)) {
        if (*line == '.' || *line == '?' || *line == '!')
            continue;
        for (px = line; *px != '\n'; px++)
            if (isupper(*px) || islower(*px))
                tlen++;
    }
    *len = tlen;
    pseq = (char *) g_calloc(1, tlen);
    if (pseq == 0)
        exit(1);
    for (py = pseq; px != line; px++)
        *py++ = *px;
    *py = '\0';
}

```

# FIGURE 40

...getseq

```

5.  py = pseq + 4;
    'len = tlen;
    rewind(fp);

    while (fgetc(line, 1024, fp) != EOF) {
        if (*line == '\n' || *line == '\t' || *line == '\r')
            continue;
10.     for (px = line, *px != '\n', px + 1) {
            if (isupper(*px))
                *py++ = *px;
            else if (islower(*px))
                *py++ = toupper(*px);
15.     if (index("ATGCU", *(py - 1))
                natgc + 1;
        }
    }
    *py++ = '\0';
    *py = '\0';
    (void) fclose(fp);
    dna = natgc * (tlen/3);
    return(pseq + 4);
20. }

```

```

25. char *
g_calloc(msg, nx, sz)
    /* program, calling routine */
    char *msg;
    int nx, sz;
    /* number and size of elements */
30. {
    char *px, *calloc();

    if ((px = calloc((unsigned)nx, (unsigned)sz)) == 0) {
        if (*msg) {
35.             fprintf(stderr, "%s: g_calloc() failed %s (n = %d, sz = %d)\n", prog, msg, nx, sz);
            exit(1);
        }
    }
    return(px);
40. }

```

g\_calloc

```

/*
 * get final jmps from dx[] or tmp file, set pp[], reset dmax=main()
 */
45. readjmps()
{
    int fd = 1;
    int siz, i0, i1;
    register i, j, xx;

    if (fp) {
        (void) fclose(fp);
        if ((fd = open(name, O_RDONLY, 0)) < 0) {
            fprintf(stderr, "%s: can't open() %s\n", prog, name);

```

readjmps

```

        while (i) {
            for (j = d[jmax] - jmp + 1; j < 0 && d[jmax] > 0; j++)
                xx = 0;

```

# FIGURE 4P

...readjumps

```

0      if (q == 0 && dx[dmax] < 0) {
1          (void) fseek(fd, dx[dmax] * sizeof(struct jmp), 0);
2          (void) read(fd, (char *)&dx[dmax], sizeof(struct jmp));
3          (void) read(fd, (char *)&dx[dmax], sizeof(dx[dmax]));
4          dx[dmax] += MAXJMP;
5      }
6      else
7          break;
8
9      }
10     if (q == JMPS) {
11         fprintf(stderr, "%s: too many gaps in alignment\n", prog);
12         cleanup(1);
13     }
14     if (q == 0) {
15         siz = dx[dmax] * n;
16         xx = dx[dmax] * x;
17         dmax += siz;
18         if (siz > 0) { /* gap in second seq */
19             pp[1][n] = siz;
20             xx += siz;
21
22             /* id = xx - yy + len - 1 */
23             /*
24              pp[1][x] = xx - dmax + len - 1;
25              papy += 1;
26              ngapy += siz;
27
28              /* ignore MAXGAP when doing endgaps */
29              siz = (siz + MAXGAP + 1) / 2; siz = MAXGAP;
30              n += 1;
31          }
32          else if (siz < 0) { /* gap in first seq */
33              pp[0][n] = siz;
34              pp[0][x] = xx;
35              papy += 1;
36              ngapy += siz;
37
38              /* ignore MAXGAP when doing endgaps */
39              siz = (siz + MAXGAP + 1) / 2; siz = MAXGAP;
40              n += 1;
41          }
42      }
43      else
44          break;
45
46      }
47
48     /* reverse the order of jumps
49     */
50     for (i = 0; i < j; i += 1, j -= 1) {
51         r = pp[0][n], pp[0][n] = pp[0][i], pp[0][i] = r;
52         r = pp[0][x], pp[0][x] = pp[0][i], pp[0][i] = r;
53     }
54     for (i = 0; i < j; i += 1, j -= 1) {
55         r = pp[1][n], pp[1][n] = pp[1][i], pp[1][i] = r;
56         r = pp[1][x], pp[1][x] = pp[1][i], pp[1][i] = r;
57     }
58
59     (void) unlink(prog);
60     n = 0;
61     offset = 0;

```

## FIGURE 4Q

writejumps

```

/*
 * write a filled jmp struct offset of the prev one (if any) nwo
 */
writejmp(ix)
{
    int      ix;
    char      *mktemp();
    if (!f) {
        if (mktemp(name) < 0) {
            fprintf(stderr, "%s: can't mktemp() '%s'\n", prog, name);
            cleanup();
        }
        if (!f) {
            fopen(name, "w") < 0) {
                fprintf(stderr, "%s: can't write '%s'\n", prog, name);
                exit(1);
            }
        }
        (void) fwrite((char *)&dx[ix] ip, sizeof(struct jmp), 1, fp);
        (void) fwrite((char *)&dx[ix] offset, sizeof(dx[ix] offset), 1, fp);
    }
}

```

# FIGURE 5

5  
 GTGCTCTCCAGGACAAGCAGGAGGNGGTGGAGCTGGTCAAGCAACATCTGTGGGCTCTG  
 GAAGTGTGCTACATCTCAGGCTTTGGTCTTGTCTTGGTTACTCACTTCTGTGTCTGATG  
 CGCTCACTGGTGAACACAGGACCAAGCTTCCAGCTCTGGACCCAGGAGCTGCCCCGGAC  
 TTTAGTTCCTTGCATCGGAGTCCCCATCCCTCCCGCCAAAGCAATATCTGTTCGATGAGC  
 10 TTCAGTGGCTACCAAGACAGCCTTTATCTGGCTTGGGCTCCCTGGTGGAGGAGATCATCTTC  
 TTCTTGGGAACCAAGGCTCTGGGCTTCTGTGTCTCATGCTGTGTCTGCATGCGAGGAAC  
 CTCTGTCTCTTCCCTTCTCTGAGTCTCTGGTGGGCTTCTGTGGCTGACTTTGGGCTTGGCT  
 GTGATCTTGCAGAACATGGCAGCCCATTTGGGTCTTCTTGGAGACTCATGATGACACCCA  
 CAGCTGACCAACCCGGGAGTGTCTATGCGAGCCACCTTTCTTCTCTTCCCCCTCAATGTC  
 15 CTGGTGGGTGCTATGGTGGCCACCTGGCGAGTGGTCTCTCTGTGGCTCTACAAAGGCATC  
 CACCTTGGCCAGATGGACCTCAGGCTGGCTGGCAACGAGAGCCGGCACTCTCGACCCGGG  
 TACTACAGGTACCGAA



### FIGURE 6

5 CAGAAACAGGCAAGCCCTTACGATCTAGCGACATTCGCTCTGAGCTTACAGAGAGAAAGC  
 CCGCTCTTTTGGAGAGCCCTTCTCTTCTGAGGAGACAAAGTTTCCGCGACAGATCAATAAAG  
 CAAAGC AAGACAGCAAGCAAGCGAGAGCTTACGACAAAGCCCTTTCATTTCAGGAGAAAGCGCC  
 AGAGAA**A**TGCTCTCCACGCAAGAGGAAAGAGAGCTTCCCGCGGCGGCAACAGAGCACTACT  
 CCTATGCTAGCTGGTACATCCATGAGCCCAAGCGGGCGGAGGAGCTCCAGGCCAGAGCGCG  
 10 AAGTGGCCCTCTTGGCACACAGCATACACCCGGCTCTGACCACGGCTTGGCTGGCTGGC  
 TGTCAATCTTTCTGCTGTCTCTCTTGGCATGTCTGTGAGCGCGCGGCAAGCTCTGTGCTGT  
 ACTGTGTGTGTGGAGGGCGCGCTCTGGCCAGGCCCGGGCAAGTGGCTGTCTGCTGTTTTTCA  
 TCGTCTCTCTGAGCTCCCTGTCTTTTGTCTCTCCCGCAGCAGGAGCGATTGGCCCTTCTGTGA  
 CTCTGCGCTCAGCAACCAGCATATCGAAAACTGAGGCTCCAAAGACGGGGCTGGAAAGA  
 TACTGGGACTGTTTATTATTATGCTGGCTCTTACTACCTCTTGGCTGGCTGTGGCACGGCTG  
 15 GGCACACAGCTGCAGACCTGCTCTGGAGCAGCTTTCTCTGGCCCACTTTGGCGCTCAGG  
 TCTGGCAGAGGGCAGAGTGTCCCGAGCTGCCAAGATCTAAAGTACTACTTCCCTGTGTGG  
 CCTCCCTGGCTCTCTCTGTGGCGCTGCAGTTCTCTAGCTTTTGTAGCTCTGTGCAGCTGG  
 TCAGAAAGCTTCAGCCCTAGGACAGAGCAAGCTTCAAGGGGCTGCAGACAACTACTCTGT  
 AGGAATATCTCAGGAACCTCCTTTCTAGCAAGGAAGCTGGGAAGCACTACACACCTCA  
 20 AGCATGGCTTCTCTGTCTGGCGCGCTCTCTCTTACAGACTGCATCTAGACTCCACAGC  
 CAGGATTCCATCTCCCGCTGAAGCTGGCTGTCTTCACTACACTGCAGAGGGCAATTT  
 AGCAGCTGGCGCTGCTCTGTCTCTGGCGCTGGTGAACCACTATCTAGAGCTCAGGGCAG  
 GGCTCACCAAGGATCTCTCTAGCTGTCTGGCGCTTTTGCAAATCTGCTCTCTGAGGACA  
 AGCAGCACTGGTGTGAGCTGTCTGAAGCAAGATCTTGGCTGTCTGAATCTGGCTACTCT  
 25 CAGCCCTTGTCTTGTCTCTCTTACTTACCTTCTCTGTCTGTGATGCTCTACTGGTGACAC  
 ACAGGACCAAGCTTTCAGCTCTCTCACTCAGGAGCTGCGCTGCACTTGAAGCTTTGCATC  
 CGAGTCCCATCTCTCTCTCAAGCAAGCATATTTCTTGCATGAGTTTCACTCTACAGAGA  
 CAGCCCTTATCTGGCTTTGGGTCTCTGTGAGAGATCATCTTTCTTCTGGGAACCCAGGG  
 CCTTGGCTTTCTGTGTGTGATGCTGTGTCTCATGCAAGGAACCTGCTGTCTTCCGTT  
 30 CCTTGGACTCTCTGTGGCCCTTCTTGGCTGACCTTTTGGCTTGGCTGTGATCTTGGAGAACA  
 TGGCAGGCCATTTGCTCTTTTCTTGTGAGACTCTGATGGACAGCCACAGCTGAGCAAGCGG  
 CAGTCTCTATGAGCAAGCAACCTTTCTTTCTTTCCCGCTCAATCTGTGTGTGGCTGGCATAG  
 TGGCAACCTGGGAGCTGTCTCTCTCTGCGCTCTTACAAAGCCATCCAGCTTGGGCAGATGG  
 AGCTCAGCCCTGTGTGCCACCGAGAGCGCGCACTCTCCAGCCCGGCTACTTACACGTACCGAA  
 35 ACTTCTTGAAGATTGAAGTCAGCCAGTGGCATCCAGGCATGACAGCCCTTCTGTCTCCCTGC  
 TCTTGGCAAGCGCAGAGGCTCTCTACCCAGGACCATGGCAGCCCCCAGGACAGGCTCAGAC  
 CAGGCGAGCAAGACGAAGCGCATGCAAGCTCTTACAGACAAAGCACTCCATGGCCAAAGGGAG  
 CTAGGCGCGGCGAGCTGGCGAGAGCTCTGGCTGGCTGTGGCTTACAGGCTGCTGTGACA  
 AGCCAAAGCTGTAGCTTTTGGCAAGCAAGCGCTCTCTTGTGTGGCAATGCTGTCCAGGCC**T**  
 40 **GA**AGGCAAGCAAGCTCAAGCCAGCTCTCACTCTTCTTACGATCTCTCTCTTACCAAC  
 CTCTCTCTCTCCCGCTCTCTCTCCAGCATCAAGCAAGCTATGAGCAAGAGCTCTCTC  
 GGATCACTCTGTCTTGGCTGGAGCTCTCTCTCACTGGAGGCTCAAGAGGCTCTGTCTCC  
 ACCCACTTGGCTATGGAGAGCCAGGACGGCTTCTTGAAGAAAAGAACTGTGCTTACGG  
 CCTTGGCTCCAGGAGCCAGTTGAGCCAGGGCAGCCACATCCAGGGCTCTCTCTAGCCCTGGC  
 45 TCTTGGCATCAGCTTTGAAGGACCTGCATGAAGCTTTCTTGGAAACACTCTCAGCCAGCT  
 CCAGCTTACGCTTGGCTTTCAAGCTGTGTGAAGCAAGCAAGGCACTTCTCAGCCCTCAG

# FIGURE 7

MSQCFAGHQTSFGATEDYSTYGSWYTHETQCGEELQTEGEVPSCHTSTETFGETHACCLASL  
 5 SHLVLLILLIAMIVERREQLWFDCEVEGPEGLIEPRAVPAAVEMVLSSLLCLLLELEDALEFL  
 TLASAPFQDQGETEAPRCWAKILGLEYYAALYYPLAACATAGHTAAHLLGSTLEWAHLGV  
 QVWQRAECPQVFKITKYYSILLASHLPLLLGLGLFLSLWYPVQLVPSESEFERTGAGSKGLQSS  
 YSEETLRKILLCPKKLGSSYHTSEHGFLSWARVCLHCTYTFQEGHLELELVLSATLTG  
 TATYQVALLLLVGVVPTTQKVEAGVTDMSTYLLAGEGIVLSEDEQEVVELVEHHLLWALE  
 10 VCYISALVLSCLLTFELVLMRSLVTHRTNLRALHRCGAALDLSPLHRSFHPCKQALFCWMS  
 FSATQTAFTICLGLLVQQTHFFELGTTALAEFLVLMFVLHGRNILLERSLESSEWTFWLTLAL  
 AVILQDMAAHWVFELETHDGHFQLTNERVLYAATFELLEPLHVLVGALVATWRVLLSALYN  
 AHHLGQMDLSLLFPRAATLDFGYTYTYRNFLEKLEVSQSHFAMTAFCSLLLOAQSLLEPRTM  
 AAFQDSLRFGEELEGMQLLQTKDEMAKGAFFGASFGFARWGLAYTLLEHNETLOVERKTA  
 15 LLEGANGAQP

Important features of the protein:

Signal peptide:

none

Transmembrane domain:

54-71

93-111

140-157

197-214

291-312

356-371

425-444

464-481

505-522

Motif name: N-glycosylation site.

8-12

Motif name: N-myristoylation site.

50-56

167-173

232-238

308-314

332-338

516-522

618-624

622-628

631-637

652-658

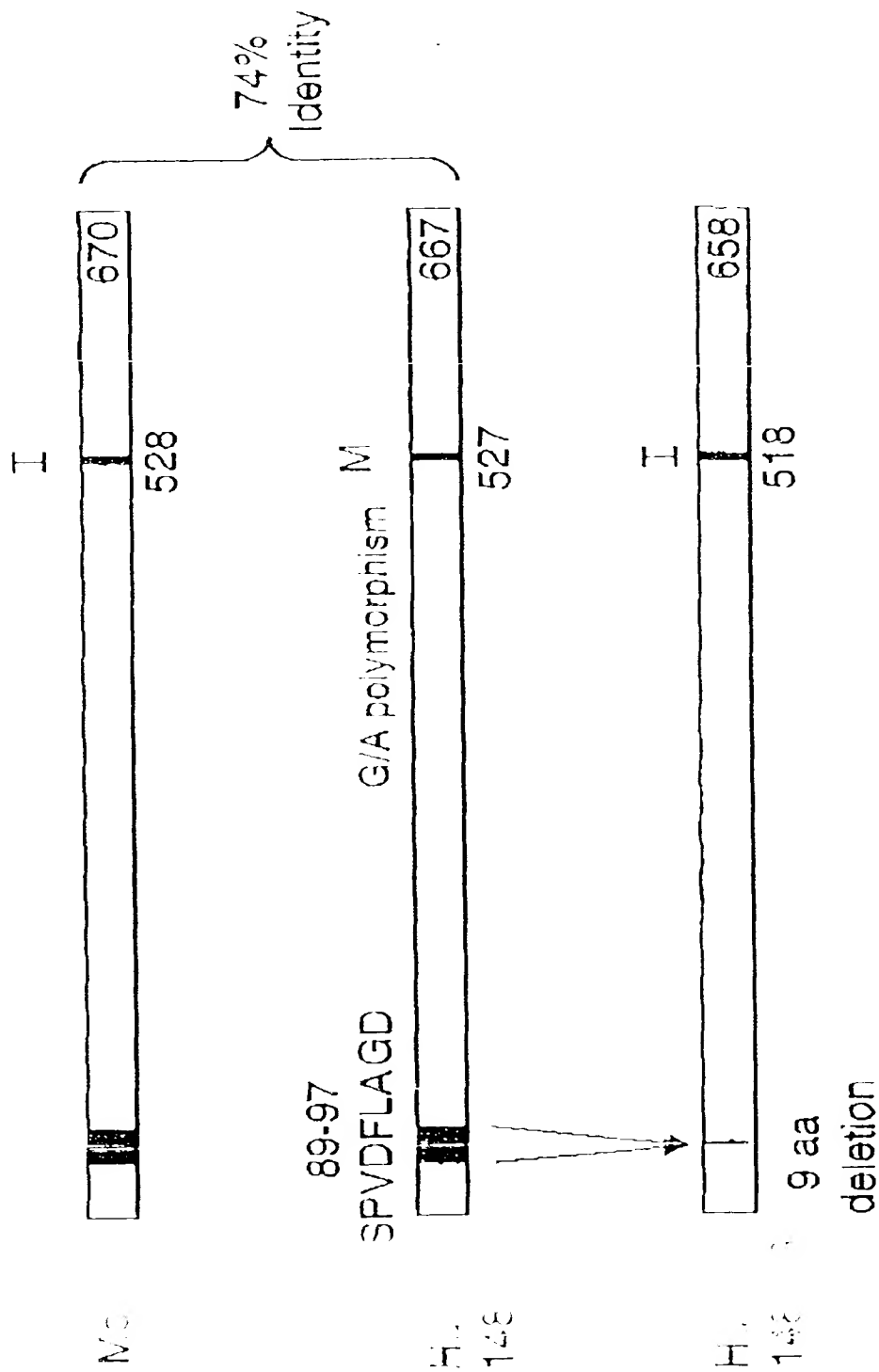
Motif name: Prokaryotic membrane lipoprotein lipid attachment

Motif name: GPI-anchored protein attachment site

157-163

# Struc6 Variant Clones

FIGURE 8



# Hydrophobicity Plot of Human Stra6

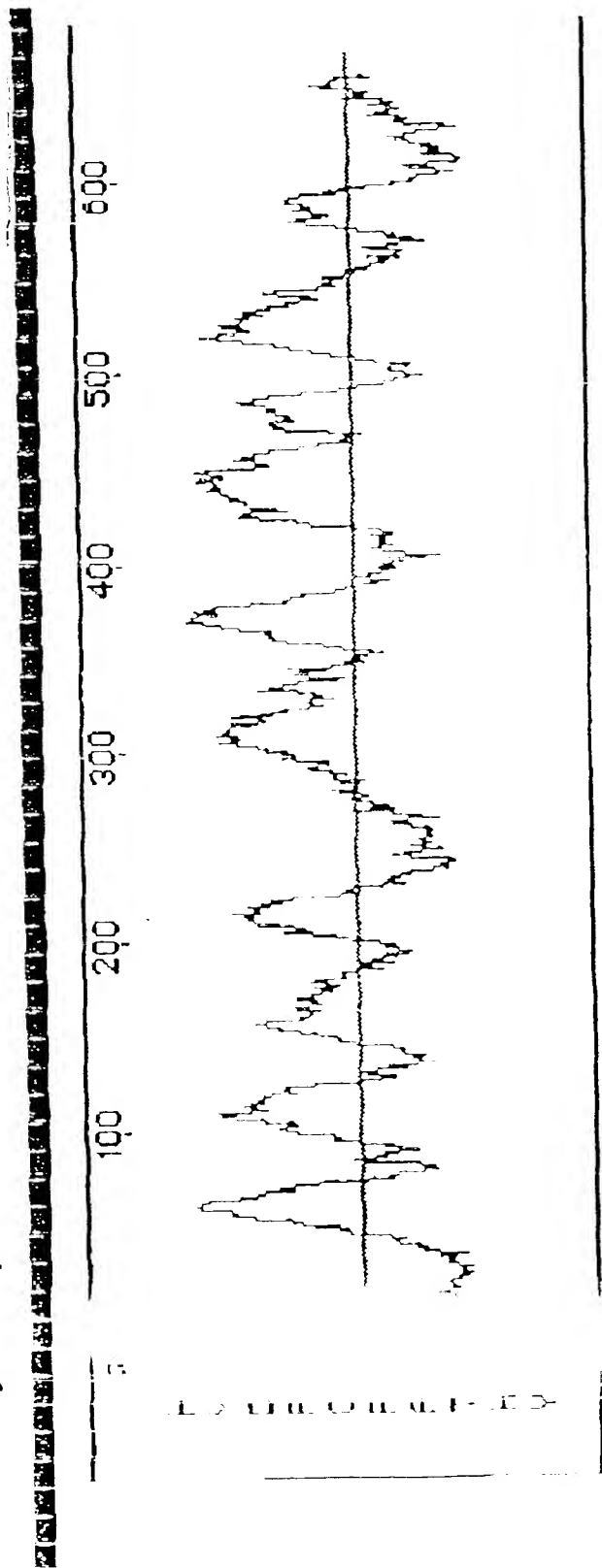


FIGURE 9

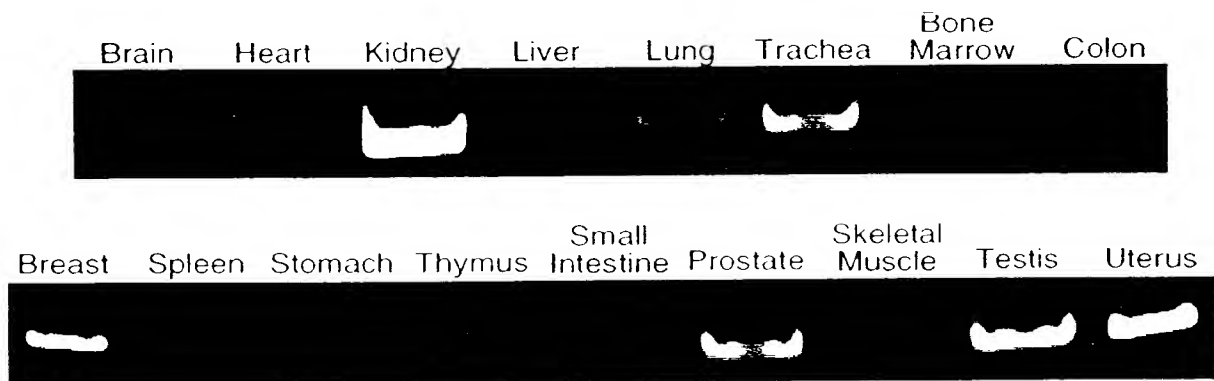
3 kb mRNA

367 Amino Acids -->50% Residues Hydrophobic

73.5 kDa Protein

9 Potential Transmembrane Domains

FIGURE 10



# Str RNA Expression in Human Colon Tumor Tissue

FIGURE 11

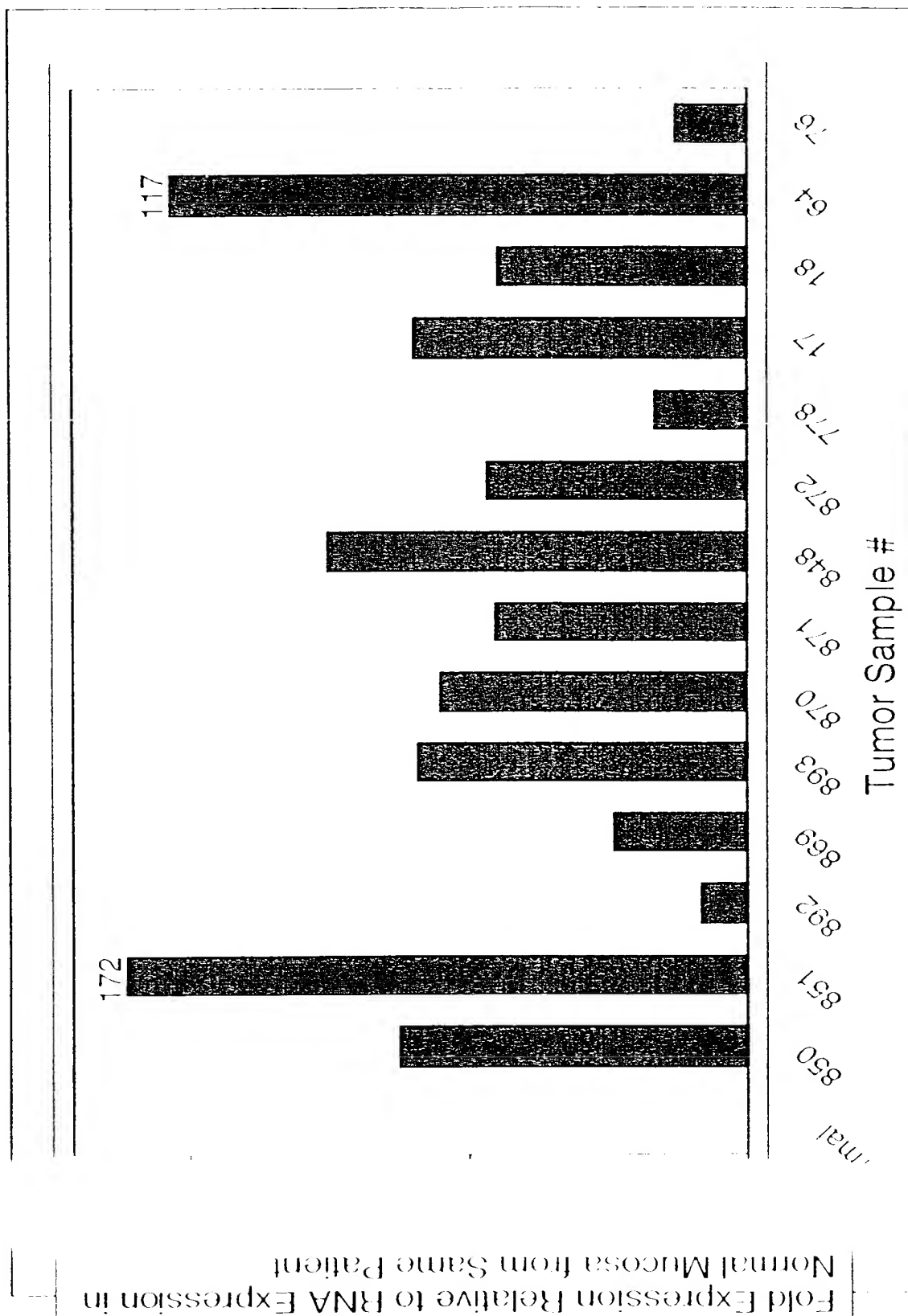


FIGURE 12A

# Stratification of Human Colon Tumor Tissue vs Normal Mucosa From the Same Patient

Taqman Product Analysis After 40 Cycles

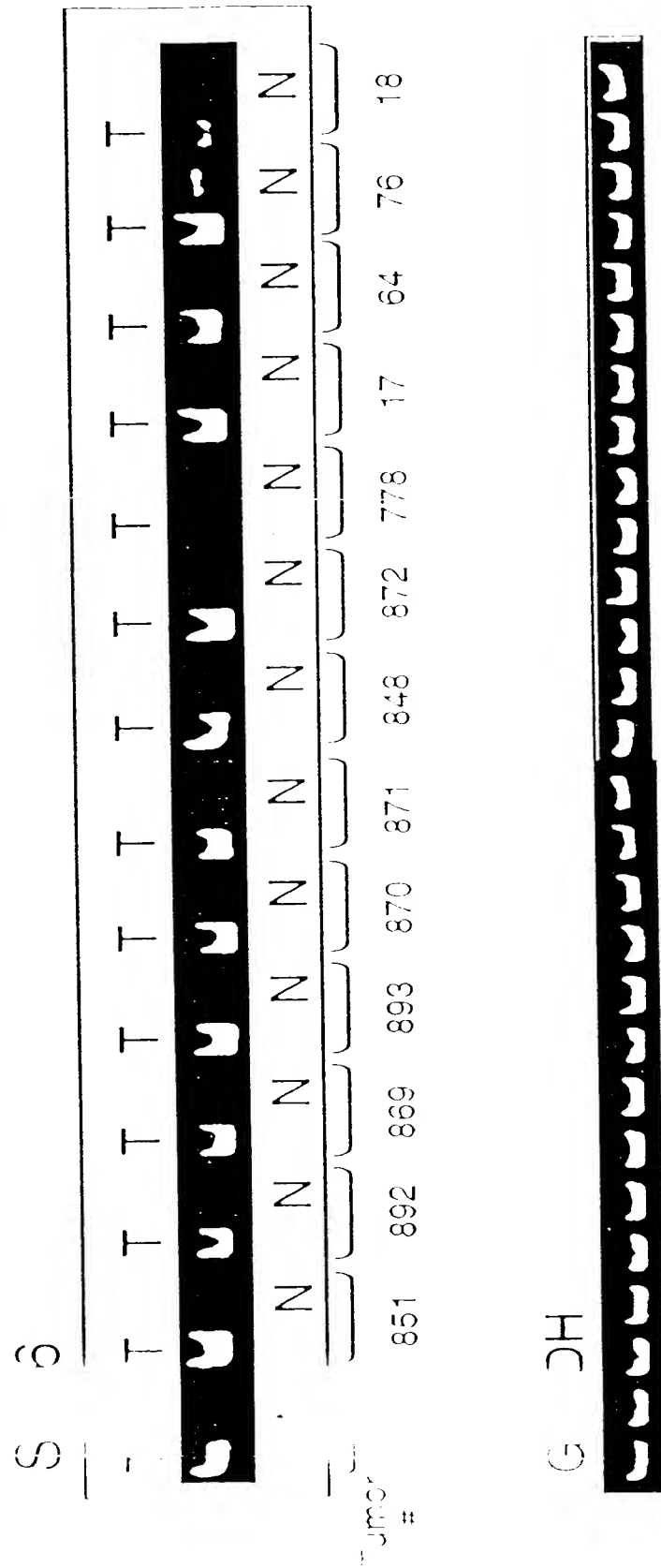


FIGURE 12B

C

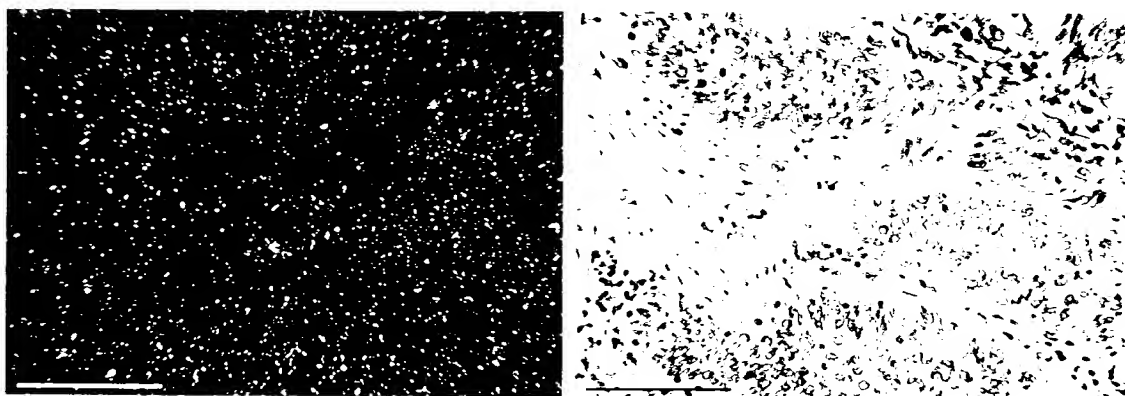
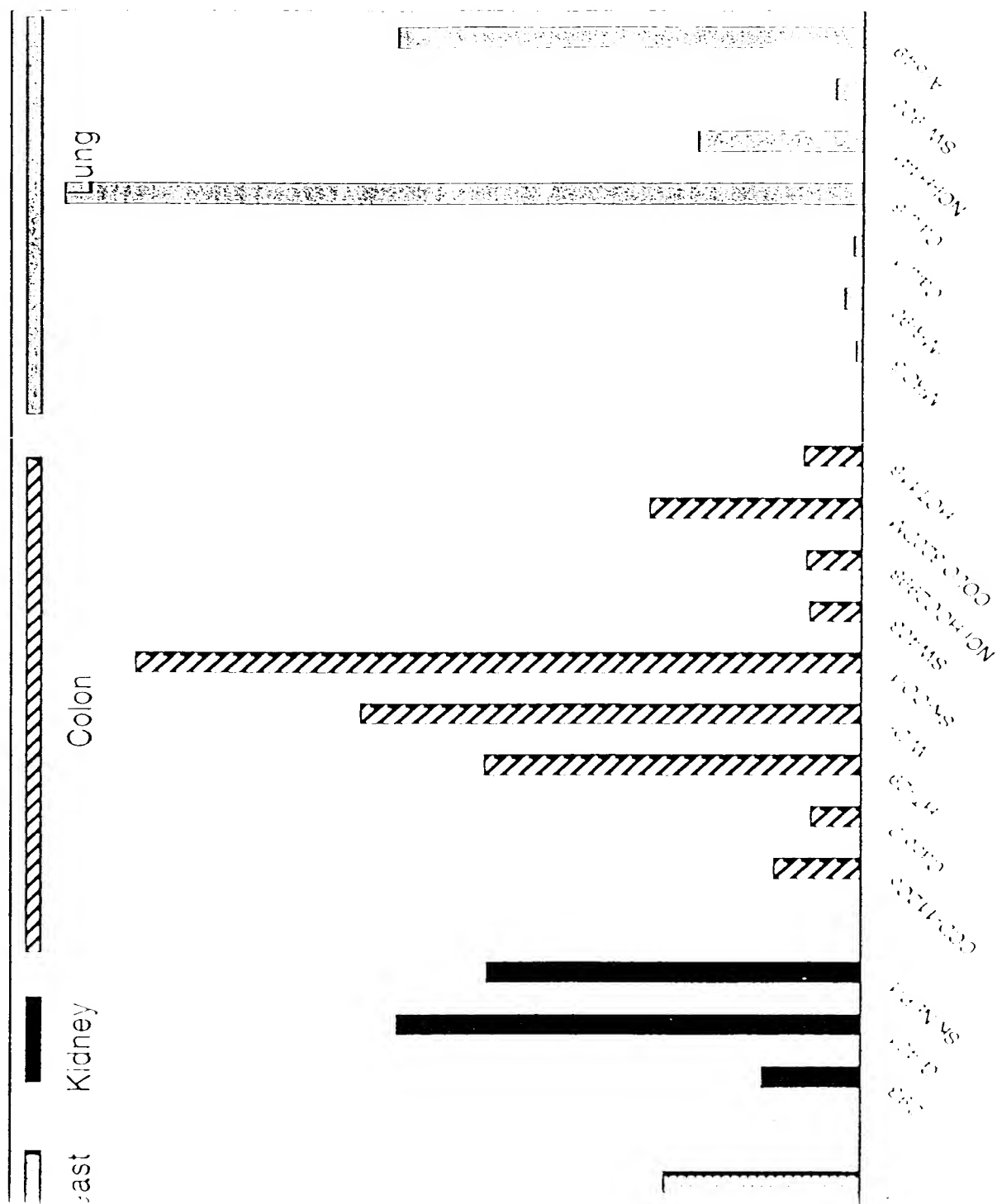


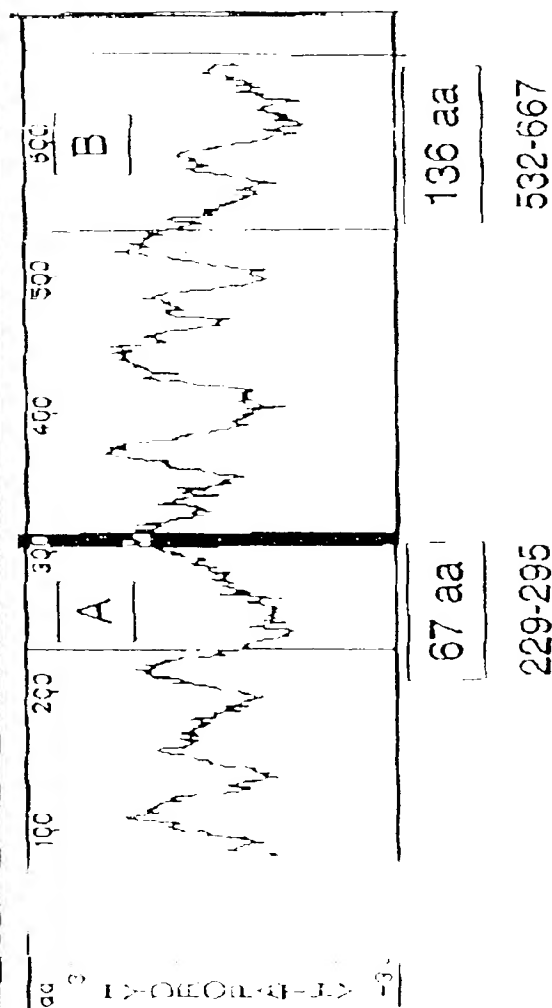


FIGURE 13



# Size Exclusion Chromatography

Poly-His Cleavable Leader at N-Terminus



Culture  
 Lane  
 mate:  
 0 µg/ml  
 g/500 ml  
 ansura

17 kD  
 9.4 kD

PBR322  
 A B

### Stra6 RNA Expression in Human Colon Carcinoma Cells +/- Retinoic Acid

TM#75 (2:28:00)

VC3 - vitamin C3 (1 μM); ATRA - all-trans-retinoic acid (1 μM).

9-cis-retinoic acid (RA)

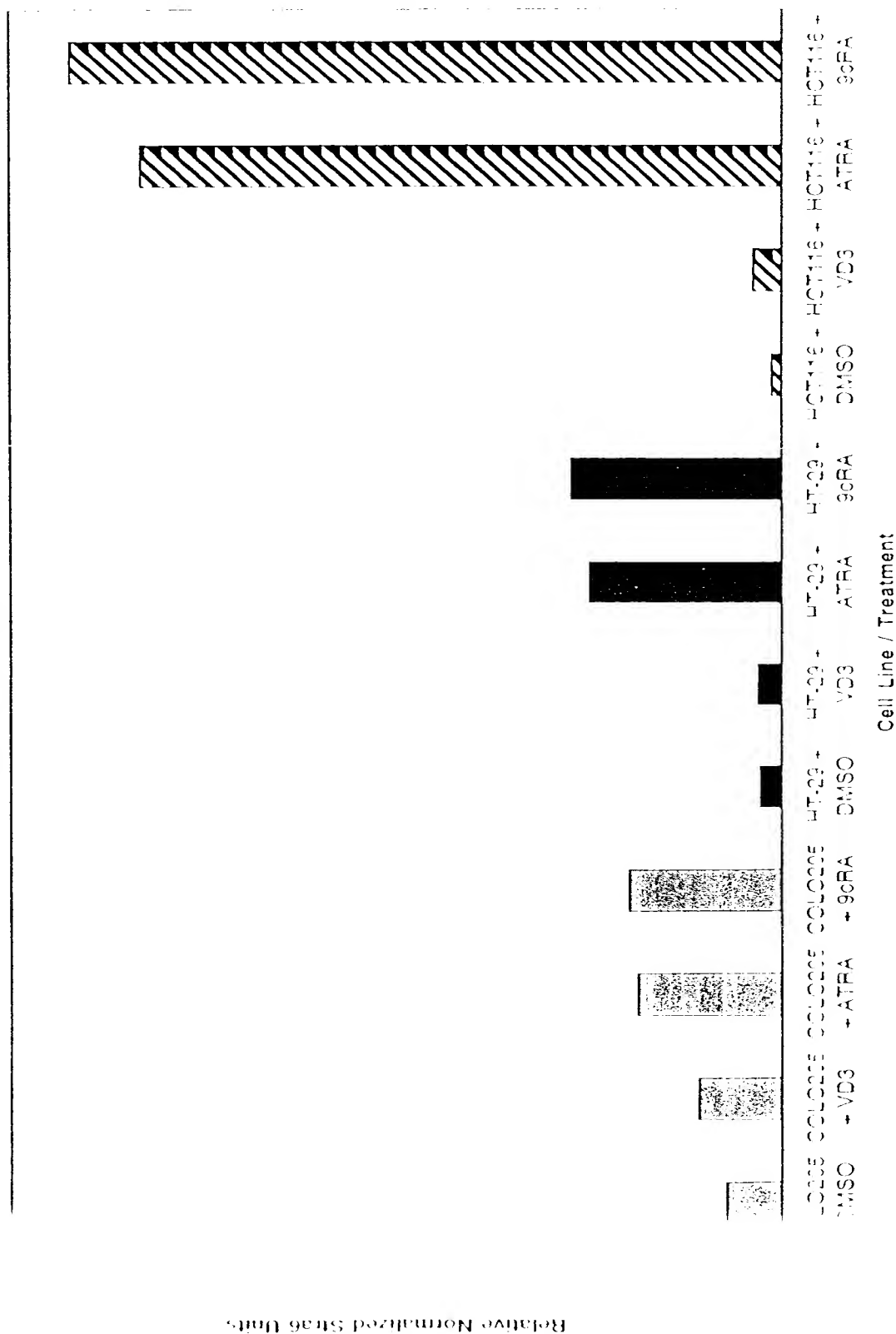


FIGURE 16

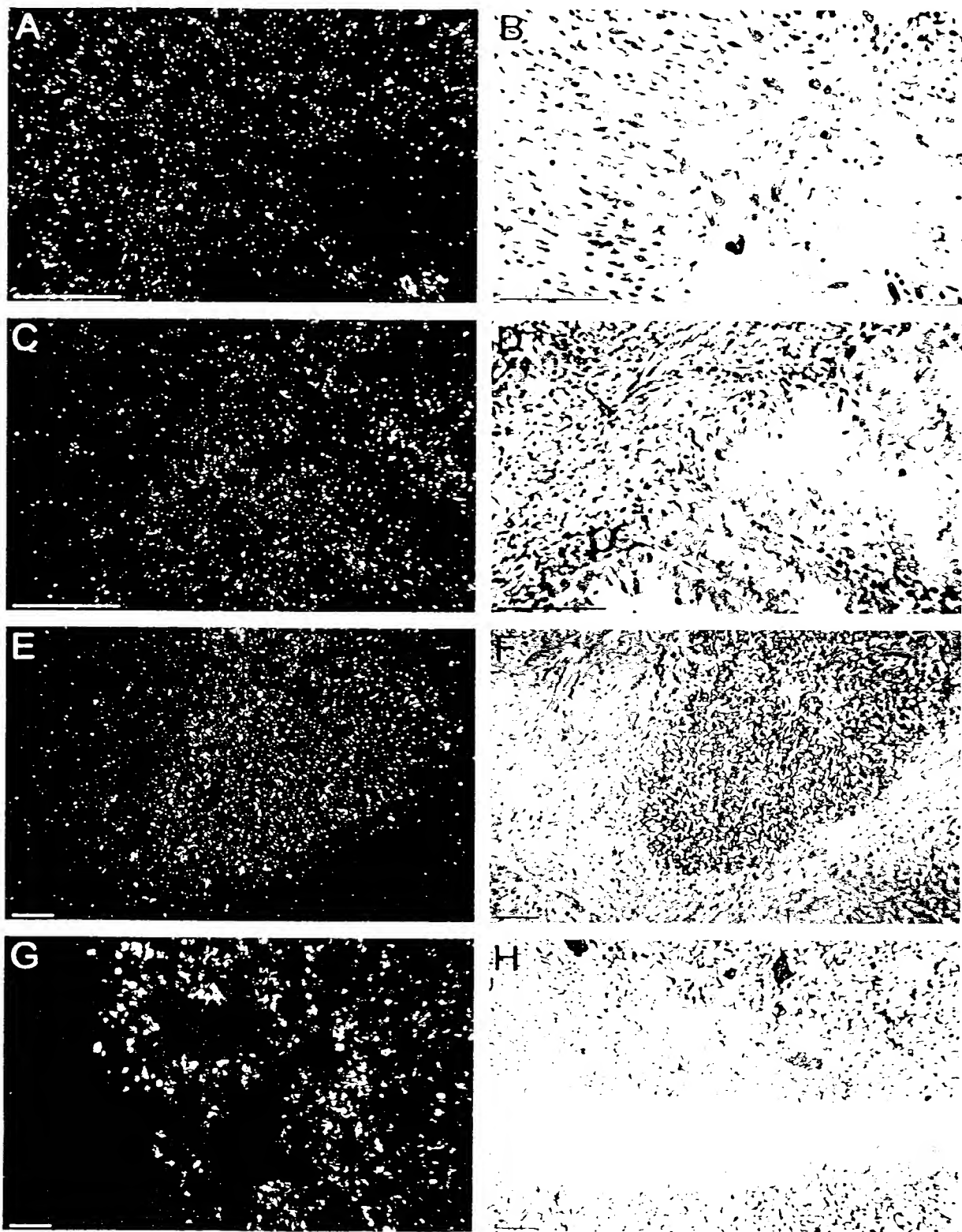


FIGURE 17

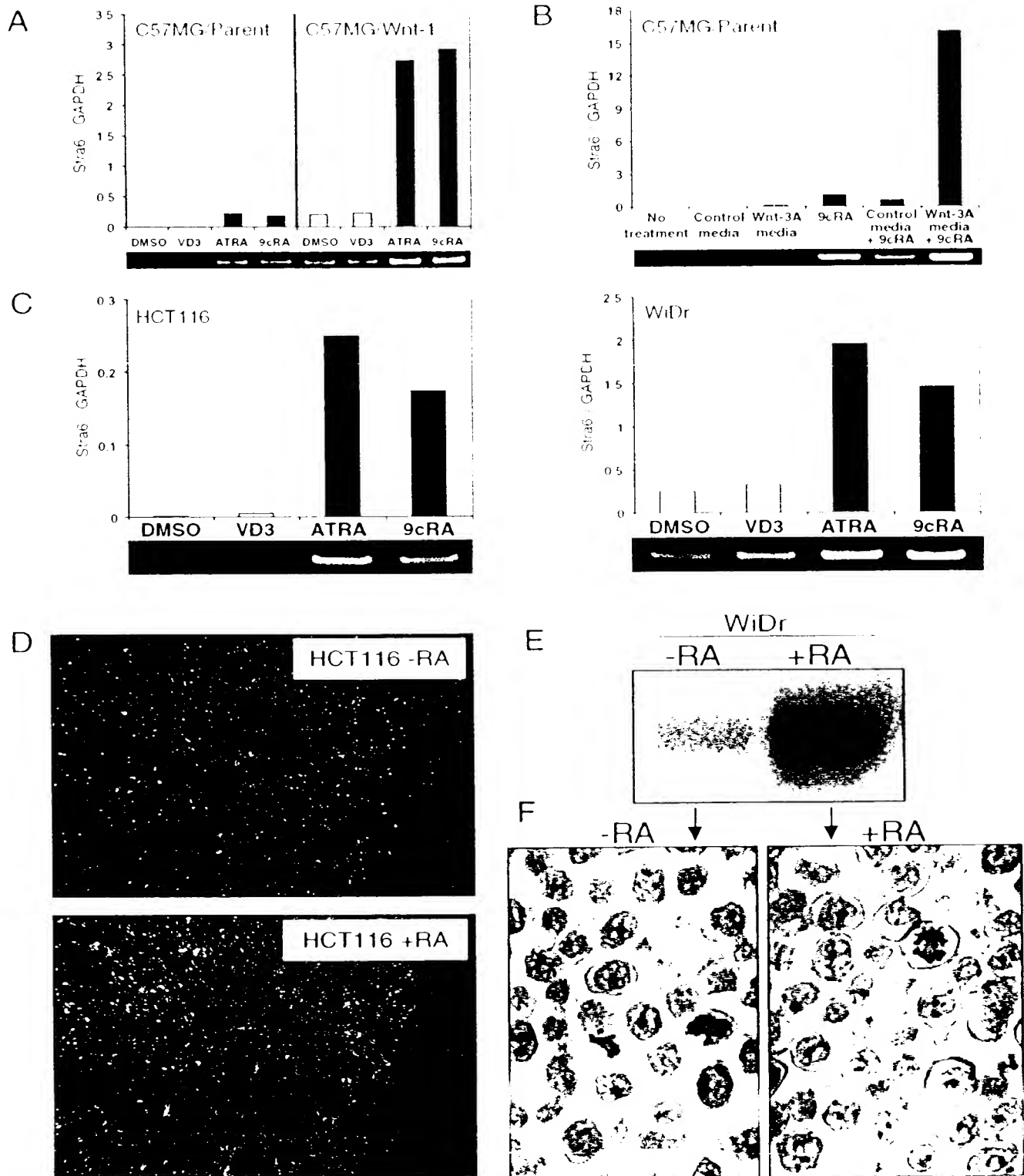


FIGURE 18

